Computer Science Department

TECHNICAL REPORT

VERIFICATION OF SEVERAL PARALLEL COORDINATION PROGRAMS BASED ON DESCRIPTIONS OF THEIR REACHABILITY SETS

by

B.D. Lubachevsky

JULY 1981

REPORT NO. 036

NEW YORK UNIVERSITY

Department of Computer Science
Courant Institute of Mathematical Sciences
251 MERCER STREET, NEW YORK, N.Y. 10012

CSD TR-036
c. 1

NYU TR 36

OCT 25 1985

Ultracomputer Note #33

VERIFICATION OF SEVERAL PARALLEL COORDINATION PROGRAMS BASED ON DESCRIPTIONS OF THEIR REACHABILITY SETS

by

B.D. Lubachevsky

JULY 1981

REPORT NO. 036

This work was supported in part by the Applied Mathematical Sciences Program of the U.S. Department of Energy under Contract No. DE-AC02-76ER03077, and in part by the National Science Foundation under Grant No. NSF-MCS79-21258.

VERIFICATION OF SEVERAL PARALLEL COORDINATION PROGRAMS BASED ON DESCRIPTIONS OF THEIR REACHABILITY SETS

by

B.D.Lubachevsky

Abstract - A method for verifying parallel programs is applied to several examples (PV-semaphore, "busy-wait" synchronization, "cessation of activity" synchronization, "readers-writers"). The graph of all program states and state transitions is represented in a special compact form independent of the number N of processing elements. This representation aids in verifying certain correctness properties that can not be easily expressed in the form "predicate(state)". In each of the above mentioned examples a special "reachability tree" is developed whose nodes are some subsets of the set of all reachable states. The root is the initial state and moving down the tree corresponds to some processors advancing their execution. In the presented examples the size of this tree is independent of N. The notion of compact program is introduced: roughly speaking a parallel program is compact if there exists a boundary, independent of N, on time required to reach any state. Examples of non-compact programs are represented.

Index Terms - correctness proof, program verification, concurrent processes, synchronization, semaphore, liveness property, mutual exclusion, deadlock.

This work was supported in part by the Applied Mathematical Sciences Program of the U.S. Department of Energy under Contract No. DE-AC02-76ER03077, and in part by the National Science Foundation under Grant No. NSF-MCS79-21258.

1. Introduction.

This paper considers verification problems typified by following (incorrect) implementation of Dijkstra's PV-semaphore.

```
COMMENT P-section

     P1:  p <- REPADD(sem,-1)
          if (p > 0) then go to P3

     P2:  REPADD (sem,1)
          go to P1

COMMENT critical section, protected by sem

     P3:  {...}

COMMENT V-section

          REPADD (sem,1)
          go to P1
```

This code is supposed to be executed by all the processing elements (PEs) of a parallel computer. In this code, sem is a public variable accessible by each PE; initially sem = 1; p is a private variable, i.e. each PE maintains its own copy of p; all PEs begin at P1; expression REPADD(sem,constant) is used for the function-with-side-effect that replaces sem by (sem + constant) and returns the new value of sem. The REPADD operation is indivisible in the sense that the result of concurently executed REPADD operations is the same as if the operations were executed in some unspecified serial order (see [5]).

It was shown in [4] (see also [5]) that after the first execution of the critical section P3, the following unacceptable race condition can occur: All the PEs are infinitely executing the loop between the statements P1 and P2, and sem $\leq$ 0, which prevents any PE from entering the critical section. Note that such "parallel bugs" are particulary hard to discover so a mechanical method for exposing them would be very helpful. Consider the following table that lists all states reachable when a 2-processor parallel computer executes the above program. In this table we characterize states by giving the values of sem and the number of PEs beginning execution of each statement P1, P2, P3. (We do

not consider states in which a PE has executed part, but not all, of a statement). For each state we also indicate the states resulting from a PE completing execution of its current statement and moving to another.

s_1 (initial state) [P1: 2 PEs, P2: no PEs, P3: no PEs; sem = 1]
  move of 1 PE from P1 to P3 leads to s_2

s_2 [P1: 1 PE, P2: no PEs, P3: 1 PE; sem = 0]
  move of 1 PE from P1 to P2 leads to s_3
  move of 1 PE from P3 to P1 leads to s_1

s_3 [P1: no PEs, P2: 1 PE, P3: 1 PE; sem = -1]
  move of 1 PE from P2 to P1 leads to s_2
  move of 1 PE from P3 to P1 leads to s_4

s_4 [P1: 1 PE, P2: 1 PE, P3: no PEs; sem = 0]
  move of 1 PE from P1 to P2 leads to s_5
  move of 1 PE from P2 to P1 leads to s_1

s_5 [P1: no PEs, P2: 2 PEs, P3: no PEs; sem = -1]
  move of 1 PE from P2 to P1 leads to s_4

Table 1.1

This type of table can be produced mechanically. In fact, a formal procedure exists and has been programmed by the author that can analyse programs like the one given above for any fixed number of PEs. We do not present a detailed description of this procedure in the present paper and only briefly outline it here. This procedure, first, creates the table of all reachable states (like table 1.1 above). Then the procedure analyses the directed graph representing all reachable states of the program and all possible transitions. For our example the nodes of the graph are s_1, s_2, s_3, s_4, s_5, and the arcs are (s_1,s_2), (s_2,s_3), (s_2,s_1), (s_3,s_2), (s_3,s_4), (s_4,s_5), (s_4,s_1), (s_5,s_1). Each of the arcs corresponds to one move listed in the above table. The race condition mentioned above appear as a strongly connected component s_4->s_5->s_4 of the subgraph generated by the predicate "there are no PEs at P3" in this graph. We will see that the well-known "finite delay property" (discussed below) also must be satisfied by the strongly connected component in order to generate a race condition.

We now show that the verificational approach started by Floyd for sequential programs and extended ([2], [7]) to concurrent programs would not expose the described above race condition. We can rewrite our REPADDs as "WITH-WHEN"s of [7]:

p <- REPADD (sem, const)

is the same as

WITH sem WHEN true DO {sem <- sem + const; p <- sem}

Here p is a private variable, sem is a public variable and const is a constant, the notation "WITH sem" represents the indivisibility of resource sem. The incorrect semaphore implementation given above is "deadlock-free" according to [7] because all processes inside WITH-WHEN sections can not be "blocked". (Blocking occures if there exists a reachable state with all WHEN conditions false. But in our example all these conditions are identically true).

The semaphore program is "improved" in [5]. But is the new program (given in section 3) correct? We have applied our verification procedure for parallel computers with various numbers of PEs and no bug was found.[1] (The time and memory requirements increase rapidly with the number of PEs.)

Can one develop for an arbitrary number of PEs a compact representation of table 1.1 containing all information about the program behavior? An example of such a representation has been given by Dijkstra [4]. While analysing a "producer-consumer" program Dijkstra introduces a set S of states of the program and analyses all possible state transitions. He shows that all transitions from a state in S yield a state in S. This analysis then is applied to verify a property of the form "predicate(state)".

[1] In [3] a similar approach was applied to analyse a parallel program, however the reachability set was developed only for 2 PEs.

But this approach is not detailed in [4]. It was not shown that all states in S are reachable, and no procedure was given to generate S. Also the possibility of using this method to investigate race condition was not mentioned, although the race condition described above was exposed in the very same paper.[2]

The present paper analyses several programs in which all execution states are represented in a special compact form for an arbitrary number of PEs. This technique aids in verifying some correctness properties of such programs, especially, those properties that can not be easily expressed in the form "predicate (state)", e.g. the absence of a race condition.[3]

[2]The author finds [4] to be a very stimulating work. However based on the existance of the race condition in the semaphore program the primitive REPADD was rejected as not appropriate for solving the mutual exclusion problem. In fact, this and many other problems can be solved using REPADD (see [5]).

[3]A few recent works such as [8] study the liveness property of parallel programs. The absence of a race condition in the semaphore example above would be such a property. However, unlike the present work [8] assumes implicitly that the number of processes is fixed, and only presents program examples of this kind. The author believes that this assumption greatly simplifies the analysis needed and as mentioned above, permits automatic verification.

2. Some definitions.

We wish to define a parallel program schema that will be called an abstract program. An abstract program represents a class of concrete parallel programs and is defined by the following:

- a directed graph G (which represents the "body" of the concrete programs). The vertices of the graph are called positions. Let $\{Pi;i=1,\ldots,k\}$ be the set of positions of G; G has two kinds of the positions: interior and exterior described below. As usual we say that Pj follows Pi, if G has an arc (Pi,Pj);

- a set C of counters, such that to each interior position P there is associated a counter c=c(P). Note that one counter can be associated with several positions. C corresponds to the public variables in the concrete program. Since the programs considered below have no private variables, C coincides with the set of all program variables[4];

- a set $F = \{f_i\}$ of replacing functions, f_i:Range(c(Pi)) -> Range(c(Pi)). Each interior position has a unique replacing function;

- a set $D = \{d_i\}$ of directing functions; d_i assigns to each value of c(Pi) a position that follows Pi. Each interior position has its own directing function. If no position follows Pi then $d_i(c(Pi))$ is the empty set.

The distinction between interior and exterior positions is that the former usually represent the control flow of a concrete program represented by the abstract program; whereas the latter represent the

[4]If there are private variables in the program, which can not be considered as counters, one can apply a "replicating code" technique to get rid of them. We give an example of how this technique works in section 4. A more detailed explanation of this technique as well as some other general issues concerning the verification problem are to follow in a future paper.

control flow of other "exterior" concrete programs. In the examples below this point will become clearer.

In the following "program" will mean an abstract program unless otherwise specified.

The execution state or simply the state of a program is the vector

$$\text{state} = (n_1,\ldots,n_k;c_1,\ldots,c_r),$$

where n_i is the current number of processing elements (PEs) at position Pi, i=1,...,k; $N = n_1+\ldots+n_k$ is the total number of PEs; c_i is as above; and r is the number of distinct counters.

We assign meaning to the given program (G,C,F,D) with a given initial state s_0 (and thus a given N) by defining a set of possible execution histories (or simply histories). Such a history is a sequence of states $s_0,s_1,\ldots$ as specified by the following (nondeterministic) procedure.

```
step <- 0; CurrentState <- s0
REPEAT
   if possible, choose a position Pi such that
    ni is positive and some position follows Pi;
   IF such a Pi does not exist
   THEN Finished <- TRUE
   ELSE
      BEGIN
         IF Pi is interior
         THEN
            BEGIN
               replace the value of the associated
                counter    c(Pi) <- fi (c(Pi));
               define the new position Pj <- di(c(Pi))
            END
         ELSE (Pi is exterior) choose a position Pj
               following Pi;
         move one PE from Pi to Pj, namely
           ni <- ni - 1,   nj <- nj + 1
           COMMENT if i=j, then ni does not change.
      END
   step <- step + 1; sstep <- CurrentState
UNTIL Finished = TRUE
```

Note that histories may be of infinite length. In the examples considered below each position has at least one successor; in such cases all histories are of infinite length.

Starting with a given initial state s_0 one can generate the set $R = R(s_0)$ of all reachable states by developing all possible histories. The reachability directed graph $\Delta=\Delta(s_0)$ is defined as follows: the node set of Δ is R and the arcs are the possible transitions, i.e. all pairs (s_i,s_{i+1}) determined by the procedure given above.

Now we wish to present our abstract program in a form closer to the usual representation of a concrete program. We will use the expression Replace-f(c) for the function-with-side-effect that replaces c by f(c) and returns the new value of c. In this new representation, the program will be written as an unordered set of statements each of which corresponds to a position of G. There are two kinds of statements: For each interior position Pi the corresponding statement is

(I) Pi: go to d_i(Replace-f_i(c(Pi)))

For each exterior position Pi the corresponding statement is

(E) Pi: go to Pj_1 or Pj_2 or...or Pj_m

where $Pj_1,\ldots,Pj_m$ follow Pi. The expression "go to empty", that occurs when no position follows Pi, should be understood as the empty statement "do nothing".

In the example programs to follow the statement order will be significant because we adopt the usual convention that if no "go to" appears in the statement corresponding to Pi, a "go to the statement corresponding to P(i+1)" is assumed. The following statement forms (all are special cases of (I) and (E)) will be used in the examples. (The expression REPADD(counter,constant) means Replace-f(counter) with f(x) = x + constant.)

(I1) Pi: counter <- constant

(I2) Pi: counter <- constant
go to Pj

(I3) Pi: if predicate(counter) then go to Pj

(I4) Pi: if predicate(counter) then go to Pj_1
go to Pj_2

(I5) Pi: REPADD(counter,constant)

(I6) Pi: REPADD(counter,constant)
go to Pj

(I7) Pi: if (predicate(REPADD(counter,constant))) then go to Pj

(I8) Pi: if (predicate(REPADD(counter,constant))) then go to Pj_1
go to Pj_2

(E1) Pi: go to Pj

(E2) Pi: go to Pj_1 or Pj_2

In subsequent examples when refering to a program we will mean a family of programs parameterized by N. Replacing functions, directing functions and the initial state (but not the graph G) may depend on N. In statements (I) and (E) we allow predicates and constants to depend on N.

3. Semaphore.

The semaphore program of [5] can be represented by the following code, where we use star (*) to represent the current position, and {...} to represent a basic block (i.e. single entry single exit) that does not access the counters. The initial state s_0 in this and subsequent examples is $(N,0,\ldots,0;c_1,\ldots,c_r)$, i.e. all N PEs begin at P1. In this example there is only one counter, sem, which has the initial value 1.

```
     P1:  {...}

COMMENT P-section

     P2:  if (sem ≤ 0) then go to *

     P3:  if (REPADD (sem,-1) ≥ 0) then go to P5

     P4:  REPADD (sem,1)
          go to P2

COMMENT critical section, protected by sem

     P5:  {...}

COMMENT V-section

     P6:  REPADD (sem,1)
          go to P1
```

As in [5] the following two assertions are to be verified about this program:

(A1) No more than 1 PE can be in the critical section at any one time.

(A2) For any time t such that no PEs are in the critical section at time t, there exists a time $t'>t$ such that some PE is in the critical section at time t'.

While analysing this (relatively simple) program we will introduce several techniques and notions used in subsequent examples.

First, we are to represent the concrete program as an abstract program of section 2. The critical section "P5: {...}" and the other section denoted {...} do not effect the analysis. Thus we can consider the following abbreviated abstract program.

```
P1:  if (sem ≤ 0) then go to *

P2:  if (REPADD (sem,-1) ≥ 0) then go to P4

P3:  REPADD (sem,1)
     go to P1

P4:  REPADD (sem,1)
     go to P1
```

Note that statements P1, P2, P3, P4 in this code are of the form (I3), (I7), (I6), and (I6), respectively.

We next show that $R(s_0) \supset$ S1 U...U S5, where Si, i=1,...,5 are the following sets (all n_i are supposed to be nonnegative integers):

$$S1 = \{s_0\} = \{s \; ; \; n_1 = N, \; sem = 1\}$$

$$S2 = \{s \; ; \; n_1 + n_2 = N, \; sem = 1\}$$

$$S3 = \{s \; ; \; n_1 + n_2 = N-1, \; n_4 = 1, \; sem = 0\}$$

$$S4 = \{s \; ; \; n_1 + n_2 + n_3 = N-1, \; n_4 = 1, \; sem = -n_3\}$$

$$S5 = \{s \; ; \; n_1 + n_2 + n_3 = N, \; n_1 \geq 1, \; n_3 \geq 1, \; sem = 1 - n_3\}$$

Clearly, $R \supset$ S1.

Consider the only state $s_0 = \{N,0,0,0;1\}$ in S1, i.e. the state in which all PEs are at P1 and the public variable sem = 1. Looking at the code we see that in state s_0 from 1 to N PEs can move from P1 to P2; such moves generate the set

$$S2' = \{s \; ; \; n_1 + n_2 = N, \; n_2 \geq 1, \; sem = 1\}$$

Note that S2 = S2′ U S1 and therefore all states in S2 are reachable, i.e. $R \supset$ S2.

Consider those states s in S2 from which PEs can move along the arc P2 -> P4, i.e. those states satisfying the condition $n_2 \geq 1$. Note that in these states at least one PE is at P2 and sem = 1. Looking at

the code we see that at most 1 PE can move from P2 to P4; such moves generate S3 and therefore all states in S3 are reachable, i.e. $R \supset S3$.

Consider those states s in S3 from which PEs can move along the arc P2 -> P3, i.e. those states satisfying the condition $n_2 \geq 1$. Note that in these states at least one PE is at P2 and sem ≤ 0. Looking at the code we see that from 1 to n_2 PEs can move from P2 to P3; such moves generate the set

$$S4' = \{s \; ; \; n_1 + n_2 + n_3 = N-1, \; n_4 = 1, \; n_3 \geq 1, \; sem = -n_3\}$$

Note that $S4 = S4' \cup S3$ and therefore all states in S4 are reachable, i.e. $R \supset S4$.

Consider those states s in S4 that have at least 1 PE at P3, i.e. those states satisfying the condition $n_3 \geq 1$. Looking at the code we see that 1 PE can move from P4 to P1; such moves generate the set S5 and therefore all states from S5 are reachable, i.e. $R \supset S5$.

This implies that each Si contains only reachable states, i.e.

$$(1) \qquad R \supset \bigcup_{i=1}^{5} Si$$

The above arguments may be abbreviated as in table 3.1.

REACHABILITY TREE

S1 $n_1 = N$, sem = 1
moves from P1 to P2 lead to S2

S2 $n_1 + n_2 = N$, sem = 1
moves from P2 to P4 lead to S3

S3 $n_1 + n_2 = N-1$, $n_4 = 1$, sem = 0
moves from P2 to P3 lead to S4

S4 $n_1 + n_2 + n_3 = N-1$, $n_4 = 1$, sem = $-n_3$
moves from P4 to P1 lead to S5, if $n_3 > 1$

S5 $n_1 + n_2 + n_3 = N$, $n_1 > 1$, $n_3 > 1$, sem = $1-n_3$

Table 3.1

We will use the REACHABILITY TREE in future examples without explicitly noting that all states represented in the table are reachable.

The word "tree" in the name of the table is due to the fact that the table can be viewed as a directed graph, which is in fact a directed tree. The nodes of this graph are the Si in the table and the arcs are pairs (Si,Sj) such that some move starting at Si "leads to Sj" according to the table. For the semaphore example the nodes are {S1,S2,S3,S4,S5} and the arcs are {(S1,S2),(S2,S3),(S3,S4),(S4,S5)}, so that the tree is simply a sequence S1->S2->S3->S4->S5. For subsequent examples the REACHABILITY TREE can be more complex.

Recall the restriction $n_3 > 1$ in moves from S4 to S5. Let us call this kind of restriction a branching condition. In the above example n_3, the only variable appearing in a branching condition, does not change in value during the move controlled by the branching condition; but sometimes a branching condition involves several variables some of which change in value during the move. In such a case the predicate must be true for the states obtained after the move.

When the number of nodes in the REACHABILITY TREE is small (as is the case for the simple semaphore program), the REACHABILITY TREE can be used directly as an instrument to observe the behavior of the program. In other examples the REACHABILITY TREE is large and

reductions are helpful. Let us demonstrate one such a reduction for our simple example.

Consider the set of sets $\Psi = \{S2,S4,S5\}$. Ψ has the property that it is a minimal subset of the set $\Phi = \{S1,S2,S3,S4,S5\}$ such that for any element Si of Φ there exists an element Sj from Ψ such that Si $\subset$ Sj. In fact we have the following inclusions:

INCLUSIONS

$$S1 \subset S2;\ S3 \subset S4$$

Moreover none of the elements from Ψ is a subset of any of the other 2 elements of Ψ. Thus we can omit S1 and S3 without losing any states.

In the REACHABILITY TREE above, for each set Si that is an element of Φ we only listed some of the possible moves. The following REACHABILITY SET DESCRIPTION (RSD) is the final product of the technique. It lists, for each Si in Ψ, all possible moves.

REACHABILITY SET DESCRIPTION

S2 $n_1 + n_2 = N$, sem = 1
moves from P1 to P2 lead to *
moves from P2 to P4 lead to S4

S4 $n_1 + n_2 + n_3 = N-1$, $n_4 = 1$, sem = $-n_3$
moves from P1 to P1 lead to *
moves from P2 to P3 lead to *
moves from P3 to P1 lead to *
moves from P4 to P1 lead to S5, if $n_3 > 1$
or lead to S2, if $n_3 = 0$

S5 $n_1 + n_2 + n_3 = N$, $n_1 > 1$, $n_3 > 1$, sem = $1-n_3$
moves from P1 to P1 lead to *
moves from P2 to P3 lead to *
moves from P3 to P1 lead to *, if $n_3 > 1$
or lead to S2, if $n_3 = 0$

Table 3.2

The phrases listed under the formulas for each Si are called directing phrases. Each directing phrase represents a class of moves "from Pi to Pj". We say that these moves are carried by the arc Pi -> Pj (of the graph G) or that this arc carries these moves.

The RSD is closed in the sense that all moves are to sets given in the description; moreover, all possible moves are represented and the initial state belongs to S2. Thus we have

(2) $R \subset S2 \cup S4 \cup S5$.

Properties (1) and (2) together give

$$R = S2 \cup S4 \cup S5$$

In subsequent examples a similar argument shows that R is the union of the sets presented in the RSD.

The above analysis has actually involved a hidden assumption that N is sufficiently large. For example when we considered those states s in S3 such that $n_2 > 0$, we tacitly assumed that $N > 1$ (or else no such s exists). For each example in this paper it is easy to see that there exists an N_0 such that for all $N \geq N_0$ all the required states do exist. In particular, for the semaphore example just discussed, one may choose $N_0 = 2$. However, the REACHABILITY TREE and the RSD tables are actually valid for all $N \geq 1$ if interpreted correctly. Although for small N some sets Si are empty and some branching conditions are unsatisfiable, no contradiction arises. This may be easily checked on a case-by-case basis.

Before analysing the semaphore program using the above RSD let us make the following general remarks.

Note that the RSD can be viewed as a directed graph Γ that differs from both the graph Δ and the REACHABILITY TREE. Namely nodes of Γ are those sets Si appearing in the RSD and arcs of Γ correspond to the

directing phrases associated with each Si. In the semaphore example Γ has 3 nodes corresponding to the 3 sets S2,S4,S5 and has 11 arcs, corresponding to the 11 directed phrases in the RSD. For example, the Γ contains arcs (S2,S2), (S5,S5), and (S5,S2) since the phrases

moves from P1 to P2 lead to *

moves from P3 to P1 lead to *, if $n_3 > 1$
or lead to S2, if $n_3 = 0$

are written under the formulas for S2 and S5.

For the reader's convenience we summarize the graphs defined thus far in table 3.3.

Graph	Node set	Elements of node set
G	{P1,...,Pk}	program positions
Δ	R	reachable states
Γ	$\{Si_1,...,Si_n$ of RSD$\}$	subsets of R

Table 3.3

Let p(s) be some predicate over the set R of nodes of Δ, i.e. a predicate over states of the program and S be a subset of R. By S$\bullet$p we denote the intersection of S with {s;p(s)=true}. By $\Delta\bullet$p we denote the subgraph of Δ induced by the predicate p, i.e. the subgraph whose node set is R$\bullet$p and whose arcs are those of Δ connecting these nodes. The following is a general problem when analysing our programs. Given a

[5]A strongly connected component c in a directed graph g is a maximal subgraph of g such that for any two (not necessarily different) nodes of c there exists a directed path in g from the first node to the second.

predicate p find all strongly connected components[5] (SCCs) in $\Delta \bullet p$. To find these SCCs one executes the following procedure called FIND SCC.

Step 1. Let $\{Si_1,\ldots,Si_n\}$ be the sets constituting the RSD (these sets are also nodes of Γ). Eliminate from Γ those Si for which corresponding sets $Si \bullet p$ are empty and eliminate all arcs adjacent to these nodes. Note that a directing phrase can be viewed as a set of pairs of states and eliminate each arc whose corresponding directing phrase does not contain a pair both of whose components satisfy p. (Note that the first elimination is a special case of the second). The graph obtained is called $\Gamma \bullet p$.

Step 2. Find all SCCs in $\Gamma \bullet p$. For each SCC L generate its description, i.e. list every $S_i \bullet p$ with its formula and for each $S_i \bullet p$ list all directing phrases (i.e. arcs in Γ) not eliminated. Let G_L be a subgraph of G, consisting of those arcs of G that carry the moves represented in the directed phrases of L (and the vertices adjacent to these arcs).

Step 3. Find all SCCs of each graph G_L constructed in step 2.

At the end of this section we will prove that if there exists a non-empty SCC K in $\Delta \bullet p$, then at least one SCC L in $\Gamma \bullet p$ will be non-empty and the corresponding G_L will contain at least one SCC M, whose arcs carry all moves of K.

It is important to note that once the RSD is given procedure FIND SCC can be automated for a large class of predicates.

Now we interrupt our general remarks and apply the RSD to verify the semaphore program. Since (A1) is equivalent to:

$-n_4 \leq 1$ for all states,

it follows immediatly from the RSD.

Note that generally any assertion like predicate(state) can be easily verified using the RSD.

To express (A2) in terms of the description, we introduce the finite delay property (FDP).

We say that a cycle in Δ satisfies the FDP if for every position Pi such that $n_i > 0$ for some state on the cycle, there is a move from Pi that corresponds to a transition from a (possibly different) state of the cycle.

Note that the FDP fails for some cycle in Δ if and only if there is a position Pi such that n_i is a positive constant during the cycle (i.e. no PE enters or leaves Pi during the cycle execution and there is at least one PE at Pi).[6]

Now (A2) can be expressed as:

there are no cycles satisfying the FDP such that $n_4 = 0$ for all states on the cycle.

First we consider the case $N > N_0 = 2$. The problem is to observe all SCCs in Δ satisfying the condition $n_4 = 0$ for all states that form the SCC. To do so let us apply the procedure FIND SCC for $p(s)=\{n_4=0\}$.

[6]This definition of the FDP appears to be weaker than the one commonly used: the usual FDP means that all PEs have finite delays. Consequently it fails if one PE stays at a position Pi during the cycle execution. However, we could prove that our (weak) FDP is equivalent to the strong FDP if we refined the notion of the program state to include the identification of which PEs (rather than how many of them) are at each point in the program. Then we could establish that given a cycle of the program execution satisfying the weak FDP, one can arrange a cycle of the program execution satisfying the strong FDP and running through the same states as the first one. Note that these cycles lie not in Δ but in its refinement corresponding to the refined notion of state just given.

Step 1 gives the graph $\Gamma \bullet p$ of the form:

$S2 \bullet p$ $n_1 + n_2 = N$, sem = 1
moves from P1 to P2 lead to *

$S5 \bullet p$ $n_1 + n_2 + n_3 = N$, $n_1 \geq 1$, $n_3 \geq 1$, sem = $1-n_3$
moves from P1 to P1 lead to *
moves from P2 to P3 lead to *
moves from P3 to P1 lead to *, if $n_3 \geq 1$
or lead to $S2 \bullet p$, if $n_3 = 0$

Note that to obtain $\Gamma \bullet p$ from the RSD table we have crossed out node S4 and all directing phrases associated with it (because $S4 \bullet p$ is empty) and the phrase

moves from P2 to P4 lead to S4

associated with node S2.

Now we proceed through step 2 and find out that $\Gamma \bullet p$ has the following two SCCs (which corresponds to the graphs L in the procedure FIND SCC).

First SCC in $\Gamma \bullet p$ (L = L1):

$S2 \bullet p$ $n_1 + n_2 = N$, sem = 1
moves from P1 to P2 lead to *

Second SCC in Γ (L = L2):

$S5 \bullet p$ $n_1 + n_2 + n_3 = N$, $n_1 \geq 1$, $n_3 \geq 1$, sem = $1-n_3$
moves from P1 to P1 lead to *
moves from P2 to P3 lead to *
moves from P3 to P1 lead to *, if $n_3 \geq 1$

Now we go to step 3 of the procedure and work with each SCC separately. L1 corresponds to the subgraph $G_{L1} \subset G$ which has no SCC.

L2 gives the subgraph $G_{L2} \subset G$ of the form P2 -> P3 -> P1 -> P1 which in turn has only one SCC, namely P1 -> P1.

Now we see that the loop P1 -> P1 can carry the only class of SCCs in Δ, namely

S5•p $n_1 + n_2 + n_3 = N$, $n_1 \geq 1$, $n_3 \geq 1$, sem = $1-n_3$
moves from P1 to P1 lead to *

But none of these SCCs satisfies the FDP, because $n_3 \geq 1$ for S5•p but there is no move from P3. This verifies (A2) for $N \geq N_0$.

For $N < N_0$ some sets Si become empty and/or some branching conditions become unsatisfiable. But this only makes our task easier by eliminating some cycles.

In subsequent examples we will not consider the case $N < N_0$ separately.

Now we return to the discussion of the procedure FIND SCC in the general case and prove that if there exists a SCC K in Δ•p, then, by applying FIND SCC, one obtains both a SCC L in Γ•p such that each node s of K lies in some node Si•p of L and a SCC M in G_L that contains arcs carrying all moves represented by directing phrases of L. To prove this fact we will actually build the graphs L and M for a given SCC K.

To build L we let $s_1 \to \ldots \to s_n \to s_1$ be a cycle in K including every state (such a cycle exists since K is a SCC). We associate with this cycle an infinite sequence $\Omega = \{Si_1 \bullet p, Si_2 \bullet p, \ldots\}$ of nodes of Γ•p as follows. State s_1 lies in some set Si_1•p, which we define to be the first term of the sequence. To the transition $s_1 \to s_2$ there is associated an arc in Γ•p (i.e. a directing phrase written under the description of Si_1•p) which was not eliminated. This arc leads to some $Si_2 \times p$, which we define to be the second term, etc. Let $\{Si_1 \bullet p, \ldots, Si_{m-1} \bullet p\}$ be the largest initial segment of Ω composed of distinct nodes of Γ•p. (Such a segment exists since Γ•p is a finite graph). Hence $Si_m \bullet p = Si_r \bullet p$ for some $1 \leq r < m$. Then except for the first

m-1 terms, Ω is an infinitively repeating cycle $Si_r \bullet p \rightarrow \ldots \rightarrow Si_m \bullet p \rightarrow Si_r \bullet p$. It is easy to see that the subgraph L′ of $\Gamma \bullet p$ induced by the set of nodes $Si_r \bullet p, Si_{r+1} \bullet p, \ldots, Si_m \bullet p$ constituting the cycle is strongly-connected. We now define L to be any SCC in $\Gamma \bullet p$, containing L′. Since by construction L′ has the property that each node s of K lies in some node $Si \bullet p$ of L′, so does L.

To build M we consider the same cycle $s_1 \rightarrow \ldots \rightarrow s_n \rightarrow s_1$ in K. Now we associate with each move $s_i \rightarrow s_{i+1}$ of this cycle an arc in the graph G that carries this move. Arcs of G may be chosen several times and we consider them with their multiplicities. The set of arcs with the set of their endpoints forms a directed graph M" that is connected and has the property that for each node, the number of input arcs is equal to the number of output arcs (counting the multiplicities). It follows from this that M" is strongly-connected. Now based on M" we form another graph M′ by eliminating multiplicities from M". M′ is a subgraph in G and is also strongly-connected since M" is. Clearly $M' \subset G_L$. Finally, we define M to be a SCC in G_L containing M′. Since the arcs of M′ carry all moves represented by directing phrases of L, so do the arcs of M.

Thus we have proved that the procedure FIND SCC is correct in the sense explained above.

4. "Busy-wait" synchronization.

In this section we will analyse a synchronisation primitive routine which was suggested by Malvin Kalos and the author of this paper and then was used in many scientific application codes developed in the "Ultracomputer Project" (see, for example, [6]). The purpose of this routine is to trap PEs until all of them complete a previous asynchronous section and then to release them for execution of another asynchronous section. Counters sem(1) and sem(2) are used to calculate

the number of PEs trapped by the routine. They work in a flip-flop manner so that sem(1) / sem(2) is used by all odd / even invocations of the routine. The private variable index is 1 (or is 2) for odd (or even) invocations. Initially sem(1) = sem(2) = 0, index = 1. (We leave to the reader the task of showing that using only one semaphore will lead to a bug).

```
COMMENT asynchronous section

 P1:  {...}

COMMENT entry to the synchronization routine

 P2:  if (REPADD (sem(index),1) > N) then go to P4

COMMENT all PES but the last one do the following

 P3:  if (sem(index) > 1) then go to *
  index <- 3-index
  go to P1

COMMENT the last PE does the following

 P4:  sem(index) <- 0
  index <- 3-index
  go to P1
```

We want to verify the following assertions about the program:

(A1) No one PE can get to the next asynchronous section P1 while some PE is still in the previous asynchronous section P1.

(A2) No one PE can be trapped in the routine P2,P3,P4 for ever.

We are not able to analyse this program as is since it has the private variable index. We can easily get rid of index by replicating the code two times. Moreover since "P1: {...}" is a basic block (i.e. single entry single exit) and does not alter sem(index), its presence does not effect our analysis and we can get rid of it too. Using both of this methods we obtain the following code:

```
P1:  if (REPADD (sem(1),1) > N) then go to P3

P2:  if (sem(1) > 1) then go to *
     go to P4

P3:  sem(1) <- 0

P4:  if (REPADD (sem(2),1) > N) then go to P6

P5:  if (sem(2) > 1) then go to *
     go to P1

P6:  sem(2) <- 0
     go to P1
```

Note that indivisibility of the operations sem(i) <- 0 is provided by our interpretation of abstract programs (see section 2). However in this particular example one can use the same operations sem(i) <- 0 without any care about their possible divisibility in the concrete program. The indivisibility follows from the RSD below. Namely a maximum of one PE can stay in P3 and P6.

Now the properties (A1), (A2) can be expressed in the form:

(A1) No one PE can get to the section P4 while some PE is still in the section P1; no one PE can get to the section P1 while some PE is still in the section P4.

(A2) For any time t such that there are PEs in the section P1, P2, P3 (odd call) at moment t there exists a time $t'>t$ such that no PEs are in the section P1, P2, P3 at moment t'; for any time t such that there are PEs in the section P4, P5, P6 (even call) at moment t there exists a time $t'>t$ such that no PEs are in the section P4, P5, P6 at moment t'.

REACHABILITY TREE

S1 $n_1 = N$, $sem(1) = sem(2) = 0$
moves from P1 to P2 lead to S2

S2 $n_1 + n_2 = N$, $1 \leq n_2 \leq N-2$,
$sem(1) = n_2$, $sem(2) = 0$
moves from P1 to P2 lead to S3, if $n_2 = N-1$

S3 $n_1 = 1$, $n_2 = N-1$, $sem(1) = N-1$, $sem(2) = 0$
moves from P1 to P3 lead to S4

S4 $n_2 = N-1$, $n_3 = 1$, $sem(1) = N$, $sem(2) = 0$
moves from P3 to P4 lead to S5

S5 $n_2 = N-1$, $n_4 = 1$, $sem(1) = sem(2) = 0$
moves from P2 to P4 lead to S6

S6 $n_2 + n_4 = N$, $n_4 > 1$, $sem(1) = sem(2) = 0$
moves from P4 to P5 lead to S7

S7 $n_2 + n_4 + n_5 = N$, $1 \leq n_5 \leq N-2$,
$sem(1) = 0$, $sem(2) = n_5$
moves from P4 to P5 lead to S8, if $n_5 = N-5$

S8 $n_2 + n_4 = 1$, $n_5 = N-1$, $sem(1) = 0$, $sem(2) = N-1$
moves from P4 to P6 lead to S4′

S4′ $n_5 = N-1$, $n_6 = 1$, $sem(1) = 0$, $sem(2) = N$
moves from P6 to P1 lead to S5′

.
.
.

Table 4.1

The rest of the tree is omitted in table 4.1. It consists of sets S5′,S6′,S7′,S8′, that are connected and their descriptions are the same as those of S5, S6, S7, S8, respectively, where symbols P(1+i), n_{1+i} are interchanged with symbols P(1+j), n_{1+j}, respectively, for i=0,1,2 and j=(i+3)mod6, and sem(1) is interchanged with symbol sem(2). Note that the description of the set S4′ can be obtained from that of S4 in the same manner.

INCLUSIONS

S1 $\subset$ S6′; S2 $\subset$ S7′; S3 $\subset$ S8′; S5 $\subset$ S6; S5′ $\subset$ S6′

REACHABILITY SET DESCRIPTION

S4 $n_2 = N-1$, $n_3 = 1$, sem(1) = N, sem(2) = 0
moves from P2 to P2 lead to *
moves from P3 to P4 lead to S6

S6 $n_2 + n_4 = N$, $n_4 \geq 1$, sem(1) = sem(2) = 0
moves from P2 to P4 lead to *
moves from P4 to P5 lead to S7

S7 $n_2 + n_4 + n_5 = N$, $1 \leq n_5 \leq N-2$,
sem(1) = 0, sem(2) = n_5
moves from P2 to P4 lead to *
moves from P4 to P5 lead to *, if $n_5 \leq N-2$
or lead to S8, if $n_5 = N-1$
moves from P5 to P5 lead to *

S8 $n_2 + n_4 = 1$, $n_5 = N-1$, sem(1) = 0, sem(2) = N-1
moves from P2 to P4 lead to *
moves from P4 to P6 lead to S4′
moves from P5 to P5 lead to *

S4′ $n_5 = N-1$, $n_6 = 1$, sem(2) = N, sem(1) = 0
moves from P5 to P5 lead to *
moves from P6 to P1 lead to S6′

S6′ $n_5 + n_1 = N$, $n_1 \geq 1$, sem(2) = sem(1) = 0
moves from P5 to P1 lead to *
moves from P1 to P2 lead to S7′

S7′ $n_5 + n_1 + n_2 = N$, $1 \leq n_2 \leq N-2$,
sem(2) = 0, sem(1) = n_2
moves from P5 to P1 lead to *
moves from P1 to P2 lead to *, if $n_2 \leq N-2$
or lead to S8′, if $n_2 = N-1$
moves from P2 to P2 lead to *

S8′ $n_5 + n_1 = 1$, $n_2 = N-1$, sem(2) = 0, sem(1) = N-1
moves from P5 to P1 lead to *
moves from P1 to P3 lead to S4
moves from P2 to P2 lead to *

Table 4.2

Since (A1) is equivalent to:

- $n_1 \times n_4 \leq 1$ for all states

it follows immediately from the description.

(A2) can be expressed in the form:

- there are no cycles satisfying the FDP such that $n_1 + n_2 + n_3 > 1$ along all states on the cycle;
and there are no cycles satisfying the FDP such that $n_4 + n_5 + n_6 > 1$ along all states on the cycle.

We will prove only the first part of the assertion as the second part will then be true by symmetry.

We apply the procedure FIND SCC of section 3 to obtain all SCCs of Δ, all states of which satisfy the predicate $p(s) = \{n_1 + n_2 + n_3 > 1\}$.

Step 1 gives the graph $\Gamma \bullet p$ as in table 4.3.

S4•p $n_2 = N-1$, $n_3 = 1$, sem(1) = N, sem(2) = 0
moves from P2 to P2 lead to *
moves from P3 to P4 lead to S6•p

S6•p $n_2 + n_4 = N$, $n_4 \geq 1$, $n_2 \geq 1$, sem(1) = sem(2) = 0
moves from P2 to P4 lead to *
moves from P4 to P5 lead to S7•p

S7•p $n_2 + n_4 + n_5 = N$, $1 \leq n_5 \leq N-2$, $n_2 \geq 1$,
sem(1) = 0, sem(2) = n_5
moves from P2 to P4 lead to *
moves from P4 to P5 lead to *, if $n_5 \leq N-2$
or lead to S8•p, if $n_5 = N-1$
moves from P5 to P5 lead to *

S8•p $n_2 + n_4 = 1$, $n_5 = N-1$, $n_2 \geq 1$, sem(1) = 0, sem(2) = N-1
moves from P2 to P4 lead to *
moves from P5 to P5 lead to *

S6′•p $n_5 + n_1 = N$, $n_1 \geq 1$, sem(2) = sem(1) = 0
moves from P5 to P1 lead to *
moves from P1 to P2 lead to S7′•p

S7′•p $n_5 + n_1 + n_2 = N$, $1 \leq n_2 \leq N-2$,
sem(2) = 0, sem(1) = n_2
moves from P5 to P1 lead to *
moves from P1 to P2 lead to *, if $n_2 \leq N-2$
or lead to S8′•p, if $n_2 = N-1$
moves from P2 to P2 lead to *

S8′•p $n_5 + n_1 = 1$, $n_2 = N-1$, $n_1 + n_2 \geq 1$,
sem(2) = 0, sem(1) = N-1
moves from P5 to P1 lead to *
moves from P1 to P3 lead to S4•p
moves from P2 to P2 lead to *

Table 4.3

Step 2 gives the following 7 SCCs in Γ•p:

1-st SCC in Γ•p:

S4•p $n_2 = N-1$, $n_3 = 1$, sem(1) = N, sem(2) = 0
moves from P2 to P2 lead to *

2-nd SCC in Γ•p:

S6•p $n_2 + n_4 = N$, $n_4 \geq 1$, $n_2 \geq 1$, sem(1) = sem(2) = 0
moves from P2 to P4 lead to *

3-rd SCC in Γ•p:

S7•p $n_2 + n_4 + n_5 = N$, $n_2 \geq 1$, $1 \leq n_5 \leq N-2$,
sem(1) = 0, sem(2) = n_5
moves from P2 to P4 lead to *
moves from P4 to P5 lead to *, if $n_5 \leq N-2$
moves from P5 to P5 lead to *

4-th SCC in Γ•p:

S8•p $n_2 + n_4 = 1$, $n_5 = N-1$, $n_2 \geq 1$, sem(1) = 0, sem(2) = N-1
moves from P2 to P4 lead to *
moves from P5 to P5 lead to *

5-th SCC in Γ•p:

S6′•p $n_5 + n_1 = N$, $n_1 \geq 1$, sem(2) = sem(1) = 0
moves from P5 to P1 lead to *

6-th SCC in Γ•p:

S7′•p $n_5 + n_1 + n_2 = N$, $1 \leq n_2 \leq N-2$,
sem(2) = 0, sem(1) = n_2
moves from P5 to P1 lead to *
moves from P1 to P2 lead to *, if $n_2 \leq N-2$
moves from P2 to P2 lead to *

7-th SCC in Γ•p:

S8′•p $n_5 + n_1 = 1$, $n_2 = N-1$, $n_1 + n_2 \geq 1$,
sem(2) = 0, sem(1) = N-1
moves from P5 to P1 lead to *
moves from P2 to P2 lead to *

Step 3 gives the following two loops as SCCs in graphs G_L for L equal to the 1-st, 3-rd, 4-th, 6-th, or 7-th SCC in Γ•p: P2 -> P2 and P5 -> P5. These loops can carry the following 5 classes of SCCs in Δ:

S4•p $n_2 = N-1$, $n_3 = 1$, $n_2 \geq 1$, sem(1) = N, sem(2) = 0
moves from P2 to P2 lead to *

S7•p $n_2 + n_4 + n_5 = N$, $1 \leq n_5 \leq N-2$, $n_2 \geq 1$,
sem(1) = 0, sem(2) = n_5
moves from P5 to P5 lead to *

S8•p $n_2 + n_4 = 1$, $n_5 = N-1$, $n_2 \geq 1$, sem(1) = 0, sem(2) = N-1
moves from P5 to P5 lead to *

S7′•p $n_5 + n_1 + n_2 = N$, $1 \leq n_2 \leq N-2$,
sem(2) = 0, sem(1) = n_2
moves from P2 to P2 lead to *

S8′•p $n_5 + n_1 = 1$, $n_2 = N-1$, $n_1 + n_2 \geq 1$,
$sem(2) = 0$, $sem(1) = N-1$
moves from P2 to P2 lead to *

But none of these SCCs satisfies the FDP, because for any SCC there is a position Pi such that n_i is a positive constant for all states of the SCC. (For example any SCC from S7•p satisfyes condition $n_2 \equiv const > 1$). This proves (A2).

5. "Cessation of activity" synchronization.

The need in this synchronization, described in [5] arises in "producer-consumer" programs. When the supply of produced data is empty and there is no hope that any PE will produce more data, all PEs are to be trapped in a special section that registers "cessation of activity". As long as data can appear, no PEs should be trapped. The comprehensive representation of this example is rather complicated and requires us to represent not only the "cessation of activity" routine but also the maintainance of upper and lower bounds for buffer of stored data, and the "overflowed buffer" synchronization routine.[7]

We will not represent here all these programs because the analysis of them together is beyond the "manual" level and requires some automatization. Hoping that such automatization is possible, we represent here an idealized version of "cessation of activity" synchronization. The first step of idealization is to consider a buffer of unlimited size. Then the program would have the following form, where b is the upper bound for the number of pieces of

[7]The "overflowed buffer" state means: all PEs have produced data but no room is left in the buffer. Logically the "overflowed buffer" situation is the same as "cessation of activity". (Interchange "consuming" with "producing", the state "buffer empty" with the state "buffer full").

information in the buffer, w is the counter for the number of PEs sitting inside the control appendix. Initially w = 0. The initial value of b is immaterial providing b > 0.

```
COMMENT distributor

     P1:  work with data (produce or consume)
          go to P2 (to insert a new piece of data)
               or to P3 (to delete one previously stored piece of data)

COMMENT insertion section

     P2:  REPADD (b,1)
          insert a piece of data into the buffer
          go to P1

COMMENT deletion section

     P3:  if (b ≤ 0) then go to P6

     P4:  if (REPADD (b,-1) ≤ -1) then go to P5
          delete a piece of data from the buffer
          go to P1

     P5:  REPADD (b,1)
          go to P6

COMMENT control appendix

     P6:  REPADD (w,1)

     P7:  if (b ≤ 0) then go to P9

COMMENT recover from control appendix

     P8:  REPADD (w,-1)
          go to P3

     P9:  if (w ≤ N-1) then go to P7

COMMENT registration of cessation of activity

     P10:  go to *
```

Unfortunately this program does not allow any compact REACHABILITY TREE: to exhaust all reachable sets one has at least to exhaust the buffer. (In section 7 we consider another example of the program that does not allow a compact REACHABILITY TREE). The next step of idealization is to eliminate counter b. Instead of b we introduce the binary flag e: e = 1 if the buffer is empty, e = 0 otherwise. Initially, e = 0. Now the program has the following form:

```
P1:  go to P2 or P3

P2:  e <- 0
     go to P1

P3:  if (e = 1) then go to P6

P4:  go to P1 or P5

P5:  e <- 1
     go to P1

P6:  REPADD (w,1)

P7:  if (e = 1) then go to P9

P8:  REPADD (w,-1)
     go to P3

P9:  if (w < N-1) then go to P7

P10: go to *
```

Note that in the code, exterior positions (P1, P4 and P10) appear for the first time in our examples. We will not discuss here the correspondence between the latter program and the original one. (It can be proved that for each history of the latter program there is a corresponding history of the former. The opposite is not true). Note, that indivisibility of the operations e <- 0 and e <- 1 is provided by our interpretation of abstract programs (see section 2). If one wants to use this code as a concrete program one has to provide the indivisibility somehow.

We want to verify the following assertions about this program:

(A1) If there is no data to work (e = 1) for any time $t > t_1$, then there exists such a time t_2 that for all $t > t_2$ all PEs are registered as free from activity ($n_{10} = N$).

(A2) If there are PEs registered as free from activity ($n_{10} > 1$) at any time t_1, then for any time $t > t_1$ there is no data to work (e = 1).

The REACHABILITY TREE, INCLUSIONS and RSD are represented in tables 5.1, 5.2.

REACHABILITY TREE

S1 $n_1 = N$, e= 0, w = 0
moves from P1 to P2 lead to S2

S2 $n_1 + n_2 = N$, e= 0, w = 0
moves from P1 to P3 lead to S3

S3 $n_1 + n_2 + n_3 = N$, e= 0, w = 0
moves from P3 to P4 lead to S4

S4 $n_1 + \ldots + n_4 = N$, e= 0, w = 0
moves from P4 to P5 lead to S5

S5 $n_1 + \ldots + n_5 = N$, e= 0, w = 0
moves from P5 to P1 lead to S6

S6 $n_1 + \ldots + n_5 = N$, $n_1 > 1$, e = 1, w = 0
moves from P1 to P2 lead to S7

S7 $n_1 + \ldots + n_5 = N$, $n_1 + n_2 > 1$, e = 1, w = 0
moves from P1 to P3 lead to S8

S8 $n_1 + \ldots + n_5 = N$, $n_1 + n_2 + n_3 > 1$, e = 1, w = 0
moves from P3 to P6 lead to S9

S9 $n_1 + \ldots + n_6 = N$, $n_1 + n_2 + n_3 + n_6 > 1$, e = 1, w = 0
moves from P2 to P1 lead to S10

S10 $n_1 + \ldots + n_6 = N$, $n_1 > 1$, e = 0, w = 0
moves from P1 to P2 lead to S11

S11 $n_1 + \ldots + n_6 = N$, $n_1 + n_2 > 1$, e = 0, w = 0
moves from P1 to P3 lead to S12

S12 $n_1 + \ldots + n_6 = N$, $n_1 + n_2 + n_3 > 1$, $e = 0$, $w = 0$
moves from P3 to P4 lead to S13

S13 $n_1 + \ldots + n_6 = N$, $n_1 + \ldots + n_4 > 1$, $e = 0$, $w = 0$
moves from P4 to P5 lead to S14

S14 $n_1 + \ldots + n_6 = N$, $n_1 + \ldots + n_5 > 1$, $e = 0$, $w = 0$
moves from P6 to P7 lead to S15

S15 $n_1 + \ldots + n_7 = N$, $n_1 + \ldots + n_5 > 1$, $e = 0$,
$n_7 < N-1$, $w = n_7$
moves from P7 to P8 lead to S16

S16 $n_1 + \ldots + n_8 = N$, $n_1 + \ldots + n_5 > 1$, $e = 0$,
$n_7 + n_8 < N-1$, $w = n_7 + n_8$
moves from P5 to P1 lead to S17

S17 $n_1 + \ldots + n_8 = N$, $n_1 > 1$, $e = 1$,
$n_7 + n_8 < N-1$, $w = n_7 + n_8$
moves from P1 to P2 lead to S18

S18 $n_1 + \ldots + n_8 = N$, $n_1 + n_2 > 1$, $e = 1$,
$n_7 + n_8 < N-1$, $w = n_7 + n_8$
moves from P1 to P3 lead to S19

S19 $n_1 + \ldots + n_8 = N$, $n_1 + n_2 + n_3 > 1$, $e = 1$,
$n_7 + n_8 < N-1$, $w = n_7 + n_8$
moves from P7 to P9 lead to S20

S20 $n_1 + \ldots + n_9 = N$, $n_1 + n_2 + n_3 > 1$, $e = 1$,
$n_7 + n_8 + n_9 < N-1$, $w = n_7 + n_8 + n_9$
moves from P3 to P6 lead to S21

S21 $n_1 + \ldots + n_9 = N$, $n_1 + n_2 + n_3 + n_6 > 1$, $e = 1$,
$n_7 + n_8 + n_9 < N-1$, $w = n_7 + n_8 + n_9$
moves from P2 to P1 lead to S22
moves from P6 to P7 lead to S27, if $n_7 + n_8 + n_9 < N-1$
or lead to S29, if $n_7 + n_8 + n_9 = N$

S22 $n_1 + \ldots + n_9 = N$, $n_1 > 1$, $e = 0$, $w = n_7 + n_8 + n_9$
$n_7 + n_8 + n_9 < N-1$
moves from P1 to P2 lead to S23

S23 $n_1 + \ldots + n_9$, $= N$, $n_1 + n_2 > 1$, $e = 0$, $w = n_7 + n_8 + n_9$,
$n_7 + n_8 + n_9 < N-1$
moves from P1 to P3 lead to S24

S24 $n_1 + \ldots + n_9 = N$, $n_1 + n_2 + n_3 > 1$, $e = 0$,
$n_7 + n_8 + n_9 < N-1$, $w = n_7 + n_8 + n_9$
moves from P3 to P4 lead to S25

S25 $n_1 + \ldots + n_9 = N$, $n_1 + \ldots + n_4 > 1$, $e = 0$,
$n_7 + n_8 + n_9 < N-1$, $w = n_7 + n_8 + n_9$
moves from P4 to P5 lead to S26

S26 $n_1 + \ldots + n_9 = N$, $n_1 + \ldots + n_5 \geq 1$, $e = 0$,
$n_7 + n_8 + n_9 \leq N-1$, $w = n_7 + n_8 + n_9$

S27 $n_1 + \ldots + n_9 = N$, $e = 1$, $n_1 + n_2 + n_3 + n_6 + n_7 \geq 1$,
$n_7 + n_8 + n_9 \leq N-1$, $w = n_7 + n_8 + n_9$
moves from P7 to P9 lead to S28

S28 $n_1 + \ldots + n_9 = N$, $e = 1$, $n_1 + n_2 + n_3 + n_6 + n_7 + n_9 \geq 1$,
$n_7 + n_8 + n_9 \leq N-1$, $w = n_7 + n_8 + n_9$

S29 $n_7 + \ldots + n_9 = N$ $e = 1$, $w = N$
moves from P9 to P10 lead to S30

S30 $n_7 + \ldots + n_{10} = N$, $e = 1$, $w = N$
moves from P8 to P3 lead to S31

S31 $n_3 + n_7 + n_8 + n_9 + n_{10} = N$, $e = 1$, $n_3 \geq 1$,
$n_7 + \ldots + n_{10} \leq N-1$, $w = n_7 + \ldots + n_{10}$
moves from P3 to P6 lead to S32

S32 $n_3 + n_6 + n_7 + n_8 + n_9 + n_{10} = N$, $e = 1$, $n_3 + n_6 \geq 1$,
$n_7 + \ldots + n_{10} \leq N-1$, $w = n_7 + \ldots + n_{10}$

INCLUSIONS

S1 $\subset \ldots \subset$ S5 $\subset$ S26;
S10 $\subset \ldots \subset$ S16 $\subset$ S26;
S6 $\subset \ldots \subset$ S9 $\subset$ S28;
S17 $\subset \ldots \subset$ S21 $\subset$ S27 $\subset$ S28;
S29 $\subset$ S30;
S31 $\subset$ S32

Table 5.1

REACHABILITY SET DESCRIPTION

S26 $n_1 + \ldots + n_9 = N$, $n_1 + \ldots + n_5 \geq 1$, $e = 0$,
$n_7 + n_8 + n_9 \leq N-1$, $w = n_7 + n_8 + n_9$
moves from P1 to P2 lead to *
moves from P1 to P3 lead to *
moves from P2 to P1 lead to *
moves from P3 to P4 lead to *
moves from P4 to P1 lead to *
moves from P4 to P5 lead to *
moves from P5 to P1 lead to S28
moves from P6 to P7 lead to *
moves from P7 to P8 lead to *
moves from P8 to P3 lead to *
moves from P9 to P7 lead to *

S28 $n_1 + \ldots + n_9 = N$, $e = 1$, $n_1 + n_2 + n_3 + n_6 + n_7 + n_9 \geq 1$,
$n_7 + n_8 + n_9 \leq N-1$, $w = n_7 + n_8 + n_9$
moves from P1 to P2 lead to *
moves from P1 to P3 lead to *
moves from P2 to P1 lead to S26
moves from P3 to P6 lead to *
moves from P4 to P1 lead to *
moves from P4 to P5 lead to *
moves from P5 to P1 lead to *
moves from P6 to P7 lead to *, if $n_7 + n_8 + n_9 \leq N-1$
or lead to S30, if $n_7 + n_8 + n_9 = N$
moves from P7 to P9 lead to *
moves from P8 to P3 lead to *
moves from P9 to P7 lead to *

S30 $n_7 + \ldots + n_{10} = N$, $e = 1$, $w = N$
moves from P7 to P9 lead to *
moves from P8 to P3 lead to S32
moves from P9 to P10 lead to *
moves from P10 to P10 lead to *

S32 $n_3 + n_6 + n_7 + n_8 + n_9 + n_{10} = N$, $e = 1$, $n_3 + n_6 \geq 1$,
$n_7 + \ldots + n_{10} \leq N-1$, $w = n_7 + \ldots + n_{10}$
moves from P3 to P6 lead to *
moves from P6 to P7 lead to *, if $n_3 + n_6 \geq 1$
or lead to S30, if $n_3 = n_6 = 0$
moves from P7 to P9 lead to *
moves from P8 to P3 lead to *
moves from P9 to P10 lead to *
moves from P10 to P10 lead to *

Table 5.2

It is easy to see that if $n_{10} = N$ happens at moment t' then it is true for all $t \geq t'$. Hence (A1) can be expressed in the form:

- there is no cycle satisfying the FDP such that the conditions $e = 1$ and $n_{10} < N$ are held for all states on cycle.

(A2) can be expressed in the form:

- for any 2 states s_1 and s_2 if $n_{10} \geq 1$ in s_1 and $e = 0$ in s_2 then there is no way from s_1 to s_2 in Δ.

To verify (A1) we use the procedure FIND SCC for $p(s) = \{e = 1 \ \& \ n_{10} \leq N-1 \}$. We do not represent here the graph $\Gamma\bullet p$ obtained after step 1 and go directly to step 2 which gives the following 2 SCCs L in $\Gamma\bullet p$.

First SCC in $\Gamma\bullet p$ (L=L1):

S28$\bullet$p $n_1 + \ldots + n_9 = N$, $e = 1$, $n_1 + n_2 + n_3 + n_6 + n_7 + n_9 \geq 1$,
$n_7 + n_8 + n_9 \leq N-1$, $n_{10} \leq N-1$, $w = n_7 + n_8 + n_9$
moves from P1 to P3 lead to *
moves from P3 to P6 lead to *
moves from P4 to P1 lead to *
moves from P4 to P5 lead to *
moves from P5 to P1 lead to *
moves from P6 to P7 lead to *, if $n_7 + n_8 + n_9 \leq N-1$
moves from P7 to P9 lead to *

Second SCC in $\Gamma\bullet p$ (L=L2):

S30$\bullet$p $n_7 + \ldots + n_{10} = N$, $n_{10} \leq N-1$, $e = 1$, $w = N$
moves from P7 to P9 lead to *
moves from P8 to P3 lead to S32$\bullet$p
moves from P9 to P10 lead to *
moves from P10 to P10 lead to *

S32$\bullet$p $n_3 + n_6 + n_7 + n_8 + n_9 + n_{10} = N$, $e = 1$, $n_3 + n_6 \geq 1$,
$n_7 + \ldots + n_{10} \leq N-1$, $w = n_7 + \ldots + n_{10}$
moves from P3 to P6 lead to *
moves from P6 to P7 lead to *, if $n_3 + n_6 \geq 1$
or lead to S30$\bullet$p, if $n_3 = n_6 = 0$
moves from P7 to P9 lead to *
moves from P8 to P3 lead to *
moves from P9 to P10 lead to *
moves from P10 to P10 lead to *

L1 gives the graph P4 -> P1 -> P3 -> P6 -> P7 -> P9, P4 -> P5 -> P1 in G_A, which has no SCC.

L2 gives the graph P8 -> P3 -> P6 -> P7 -> P9 -> P10 -> P10, which in turn on step 3 of the procedure gives the only loop M, of the form P10 -> P10. This loop can only carry the following two classes of SCCs in $\Delta\bullet p$:

$S30\bullet p$ $n_7 + \ldots + n_{10} = N$, $n_{10} \leq N-1$, $e = 1$, $w = N$
moves from P10 to P10 lead to *

$S32\bullet p$ $n_3 + n_6 + n_7 + n_8 + n_9 + n_{10} = N$, $e = 1$, $n_3 + n_6 \geq 1$,
$n_7 + \ldots + n_{10} \leq N-1$, $w = n_7 + \ldots + n_{10}$
moves from P10 to P10 lead to *

But none of these SCCs satisfies the FDP, because there is always a position which contains a positive number of "sleeping" PEs. This proves (A1).

(A2) can be seen from the RSD in the following way. The property $n_{10} > 1$ can be true only for some states in S30 and S32. But neither of these sets has a state with $e = 0$ and all transitions from a state in S30 U S32 yield a state in S30 U S32.

So the program satisfies (A2).

6. Readers and writers.

The simplest variant of the program from [5] is taken here. Initially the counter sem is equal to N.

```
 P1:  {...}

COMMENT distributor

 P2:  go to P3 (to read) or go to P8 (to write)

COMMENT reader

 P3:  if(sem < 0) then go to *

 P4:  if( REPADD (sem,-1) > 0) then go to P6

 P5:  REPADD (sem,1)
  go to P3

COMMENT critical section, protected by sem

 P6:  {...}

 P7:  REPADD (sem, 1)
  go to P1

COMMENT writer

 P8:  if(sem < N-1) then go to *

 P9:  if( REPADD (sem, -N) > 0) then go to P11

 P10:  REPADD (sem,N)
  go to P8

COMMENT critical section, protected by sem

 P11:  {...}

 P12:  REPADD (sem, N)
   go to P1
```

We want to verify the following assertions about the program:

(A1) No more than 1 writer can write, i.e. stay in P11.

(A2) While the writer is writing no readers can read, i.e. stay in P6.

(A3) While any positive number of readers are reading no one writer can write.

(A4) For any time t such that no PEs are in the critical section at time t, there exists a time $t'>t$ such that some PE is in the critical section at time t'.

We can rewrite this program in the form of the following abstract program.

```
P1:  go to P2 or to P6

P2:  if(sem ≤ 0) then go to *

P3:  if( REPADD (sem,-1) ≥ 0) then go to P5

P4:  REPADD (sem,1)
     go to P2

P5:  REPADD (sem, 1)
     go to P1

P6:  if(sem ≤ N-1) then go to *

P7:  if( REPADD (sem, -N) ≥ 0) then go to P9

P8:  REPADD (sem,N)
     go to P6

P9:  REPADD (sem, N)
     go to P1
```

The REACHABILITY TREE, INCLUSIONS and RSD are represented in tables 6.1, 6.2.

REACHABILITY TREE

S1 $n_1 = N$, sem = N
moves from P1 to P2 lead to S2

S2 $n_1 + n_2 = N$, sem = N
moves from P2 to P3 lead to S3

S3 $n_1 + n_2 + n_3 = N$, sem = N
moves from P1 to P6 lead to S4

S4 $n_1 + n_2 + n_3 + n_6 = N$, sem = N
moves from P6 to P7 lead to S5

S5 $n_1 + n_2 + n_3 + n_6 + n_7 = N$, sem = N
moves from P3 to P5 lead to S6
moves from P7 to P9 lead to S11

S6 $n_1 + n_2 + n_3 + n_5 + n_6 + n_7 = N$,
$n_5 \geq 1$, sem = $N - n_5$
moves from P7 to P8 lead to S7

S7 $n_1 + n_2 + n_3 + n_5 + n_6 + n_7 + n_8 = N$,
$n_5 \geq 1$, $n_8 \geq 1$, sem = $N \times (1 - n_8) - n_5$
moves from P5 to P1 lead to S8

S8 $n_1 + n_2 + n_3 + n_5 + n_6 + n_7 + n_8 = N$,
$n_1 + n_5 \geq 1$, $n_8 \geq 1$, sem = $N \times (1 - n_8) - n_5$
moves from P1 to P2 lead to S9

S9 $n_1 + n_2 + n_3 + n_5 + n_6 + n_7 + n_8 = N$,
$n_1 + n_2 + n_5 \geq 1$, $n_8 \geq 1$,
$sem = N \times (1 - n_8) - n_5$
moves from P1 to P6 lead to S10

S10 $n_1 + n_2 + n_3 + n_5 + n_6 + n_7 + n_8 = N$,
$n_1 + n_2 + n_5 + n_6 \geq 1$, $n_8 \geq 1$,
$sem = N \times (1 - n_8) - n_5$

S11 $n_1 + n_2 + n_3 + n_6 + n_7 + n_9 = N$,
$n_9 = 1$, $sem = 0$
moves from P3 to P4 lead to S12

S12 $n_1 + \ldots + n_4 + n_6 + n_7 + n_9 = N$,
$n_9 = 1$, $sem = -n_4$
moves from P7 to P8 lead to S13

S13 $n_1 + \ldots + n_4 + n_6 + \ldots + n_9 = N$,
$n_9 = 1$, $sem = -N \times n_8 - n_4$
moves from P9 to P1 lead to S14, if $n_8 = 0$, and $n_4 \geq 1$
or lead to S23, if $n_8 \geq 1$

S14 $n_1 + \ldots + n_4 + n_6 + n_7 = N$
$n_4 \geq 1$, $n_1 \geq 1$,
$sem = N - n_4$
moves from P1 to P2 lead to S15

S15 $n_1 + \ldots + n_4 + n_6 + n_7 = N$
$n_4 \geq 1$, $n_1 + n_2 \geq 1$,
$sem = N - n_4$
moves from P1 to P6 lead to S16

S16 $n_1 + \ldots + n_4 + n_6 + n_7 = N$
$n_4 \geq 1$, $n_1 + n_2 + n_6 \geq 1$,
$sem = N - n_4$
moves from P2 to P3 lead to S17

S17 $n_1 + \ldots + n_4 + n_6 + n_7 = N$
$n_4 \geq 1$, $n_1 + n_2 + n_3 + n_6 \geq 1$,
$sem = N - n_4$
moves from P3 to P5 lead to S18

S18 $n_1 + \ldots + n_7 = N$
$n_4 \geq 1$, $n_1 + n_2 + n_3 + n_5 + n_6 \geq 1$
$sem = N - n_4 - n_5$
moves from P7 to P8 lead to S19

S19 $n_1 + \ldots + n_8 = N$, $n_8 \geq 1$,
$n_4 \geq 1$, $n_1 + n_2 + n_3 + n_5 + n_6 \geq 1$,
$sem = N \times (1 - n_8) - n_4 - n_5$
moves from P3 to P4 lead to S20
moves from P4 to P2 lead to S22

S20 $n_1 + \ldots + n_8 = N$, $n_8 \geq 1$,
$n_4 \geq 2$,
$sem = N \times (1 - n_8) - n_4 - n_5$
moves from P4 to P2 lead to S21

S21 $n_1 + \ldots + n_8 = N$, $n_8 \geq 1$,
$n_2 + n_4 \geq 2$,
$sem = N \times (1 - n_8) - n_4 - n_5$

S22 $n_1 + \ldots + n_8 = N$, $n_8 \geq 1$,
$n_2 \geq 1$, $n_1 + n_2 + n_3 + n_5 + n_6 \geq 2$
$sem = N \times (1 - n_8) - n_4 - n_5$

S23 $n_1 + \ldots + n_4 + n_6 + n_7 + n_8 = N$, $n_8 \geq 1$,
$n_1 \geq 1$,
$sem = N \times (1 - n_8) - n_4$
moves from P1 to P2 lead to S24

S24 $n_1 + \ldots + n_4 + n_6 + n_7 + n_8 = N$, $n_8 \geq 1$,
$n_1 + n_2 \geq 1$,
$sem = N \times (1 - n_8) - n_4$
moves from P1 to P6 lead to S25

S25 $n_1 + \ldots + n_4 + n_6 + n_7 + n_8 = N$, $n_8 \geq 1$,
$n_1 + n_2 + n_6 \geq 1$,
$sem = N \times (1 - n_8) - n_4$

Table 6.1

INCLUSIONS

$S1 \subset \ldots \subset S5$;
$S7 \subset \ldots \subset S10$;
$S11 \subset S12 \subset S13$;
$S14 \subset \ldots \subset S18$;
$S20 \subset S21$;
$S23 \subset S24 \subset S25$

REACHABILITY SET DESCRIPTION

S5 $n_1 + n_2 + n_3 + n_6 + n_7 = N$, sem = N
moves from P1 to P2 lead to *
moves from P1 to P6 lead to *
moves from P2 to P3 lead to *
moves from P3 to P5 lead to S6
moves from P6 to P7 lead to *
moves from P7 to P9 lead to S13

S6 $n_1 + n_2 + n_3 + n_5 + n_6 + n_7 = N$,
$n_5 \geq 1$, sem = $N - n_5$
moves from P1 to P2 lead to *
moves from P1 to P6 lead to *
moves from P2 to P3 lead to *
moves from P3 to P5 lead to *
moves from P5 to P1 lead to *, if $n_5 \geq 1$
or lead to S5, if $n_5 = 0$
moves from P7 to P8 lead to S10

S10 $n_1 + n_2 + n_3 + n_5 + n_6 + n_7 + n_8 = N$,
$n_1 + n_2 + n_5 + n_6 \geq 1$, $n_8 \geq 1$,
sem = $N \times (1 - n_8) - n_4 - n_5$
moves from P1 to P2 lead to *
moves from P1 to P6 lead to *
moves from P2 to P2 lead to *
moves from P3 to P4 lead to S19
moves from P5 to P1 lead to *
moves from P6 to P6 lead to *
moves from P7 to P8 lead to *
moves from P8 to P6 lead to *, if $n_8 \geq 1$
or lead to S6, if $n_8 = 0$, $n_5 \geq 1$
or lead to S5, if $n_5 = n_8 = 0$

S13 $n_1 + \ldots + n_4 + n_6 + \ldots + n_9 = N$,
$n_9 = 1$, $sem = -N \times n_8 - n_4$
moves from P1 to P2 lead to *
moves from P1 to P6 lead to *
moves from P2 to P2 lead to *
moves from P3 to P4 lead to S19
moves from P4 to P2 lead to *
moves from P6 to P6 lead to *
moves from P7 to P8 lead to *
moves from P8 to P6 lead to *
moves from P9 to P1 lead to S18, if $n_8 = 0$, and $n_4 \geq 1$
or lead to S25, if $n_8 \geq 1$
or lead to S5, if $n_8 = n_4 = 0$

S18 $n_1 + \ldots + n_7 = N$
$n_4 \geq 1$, $n_1 + n_2 + n_3 + n_5 + n_6 \geq 1$
$sem = N - n_4 - n_5$
moves from P1 to P2 lead to *
moves from P1 to P6 lead to *
moves from P2 to P3 lead to *
moves from P3 to P5 lead to *
moves from P4 to P2 lead to *, if $n_4 \geq 1$
or lead to S6, if $n_4 = 0$, $n_5 \geq 1$
or lead to S5, if $n_4 = n_5 = 0$
moves from P5 to P1 lead to *
moves from P6 to P6 lead to *

S19 $n_1 + \ldots + n_8 = N$, $n_8 \geq 1$,
$n_4 \geq 1$, $n_1 + n_2 + n_3 + n_5 + n_6 \geq 1$,
$sem = N \times (1 - n_8) - n_4 - n_5$
moves from P1 to P2 lead to *
moves from P1 to P6 lead to *
moves from P2 to P2 lead to *
moves from P3 to P4 lead to S21
moves from P4 to P2 lead to S22
moves from P5 to P1 lead to *
moves from P6 to P6 lead to *
moves from P7 to P8 lead to *
moves from P8 to P6 lead to *, if $n_8 \geq 1$
or lead to S18, if $n_8 = 0$

S21 $n_1 + \ldots + n_8 = N$, $n_8 \geq 1$,
$n_2 + n_4 \geq 2$,
$sem = N \times (1 - n_8) - n_4 - n_5$
moves from P1 to P2 lead to *
moves from P1 to P6 lead to *
moves from P2 to P2 lead to *
moves from P3 to P4 lead to *
moves from P4 to P2 lead to *
moves from P5 to P1 lead to *
moves from P6 to P6 lead to *
moves from P7 to P8 lead to *
moves from P8 to P6 lead to *, if $n_8 \geq 1$
or lead to S18, if $n_8 = 0$, $n_4 \geq 1$
or lead to S6, if $n_8 = n_4 = 0$, $n_5 \geq 1$
or lead to S5, if $n_4 = n_5 = n_8 = 0$

S22 $n_1 + \ldots + n_8 = N$, $n_8 \geq 1$,
$n_2 \geq 1$, $n_1 + n_2 + n_3 + n_5 + n_6 \geq 2$
$sem = N \times (1 - n_8) - n_4 - n_5$
moves from P1 to P2 lead to *
moves from P1 to P6 lead to *
moves from P2 to P2 lead to *
moves from P3 to P4 lead to S21
moves from P4 to P2 lead to *
moves from P5 to P1 lead to *
moves from P6 to P6 lead to *
moves from P7 to P8 lead to *
moves from P8 to P6 lead to *, if $n_8 \geq 1$
or lead to S18, if $n_8 = 0$, $n_4 \geq 1$
or lead to S6, if $n_8 = n_4 = 0$, $n_5 \geq 1$
or lead to S5, if $n_4 = n_5 = n_8 = 0$

S25 $n_1 + \ldots + n_4 + n_6 + n_7 + n_8 = N$, $n_8 \geq 1$,
$n_1 + n_2 + n_6 \geq 1$,
$sem = N \times (1 - n_8) - n_4$
moves from P1 to P2 lead to *
moves from P1 to P6 lead to *
moves from P2 to P2 lead to *
moves from P3 to P4 lead to *
moves from P4 to P2 lead to *
moves from P5 to P1 lead to *
moves from P6 to P6 lead to *
moves from P7 to P8 lead to *
moves from P8 to P6 lead to *, if $n_8 \geq 1$
or lead to S18, if $n_8 = 0$, $n_4 \geq 1$
or lead to S5, if $n_4 = n_8 = 0$

Table 6.2

Since properties (A1), (A2), (A3) together can be expressed as

- $n_9 \leq 1$, $n_5 \times n_9 = 0$ for all states,

they follow immediately from the description.

(A4) can be expressed in the form:

- there are no cycles satisfying the FDP such that $n_5 = n_9 = 0$ along all the states on the cycle.

We apply the procedure FIND SCC for $p(s) = \{n_5 = n_9 = 0\}$ and find out that there are only 7 SCCs L in $\Gamma\bullet p$. These correspond to the single nodes S5•p, S10•p, S18•p, S19•p, S21•p, S22•p, S25•p, looping to themselves. It is easy to verify that only the following two loops can carry moves in the SCC under question. These moves are

moves from P2 to P2 lead to *

moves from P6 to P6 lead to *

Now (A4) follows from the fact that none of the sets S10•p, S18•p, S19•p, S21•p, S22•p, S25•p has a state, satisfying condition $n_2 + n_6 = N$ that would have been the only possibility to arrange the cycle, satisfying the FDP.

7. Conclusion.

To build a compact RSD we reduce a REACHABILITY TREE that in turn must be compact. The first "producer-consumer" program presented in section 5 (which we henceforth refer to as PC) does not allow a compact REACHABILITY TREE for the obvious reason that it has an unbounded

counter: For any fixed number of PEs, N, any REACHABILITY TREE for PC is infinite. In this section we present a less trivial example that can not possess any compact REACHABILITY TREE independent of N. However unlike the PC for any given N the REACHABILITY TREE is finite. This example possesses the RSD consisting of 2 sets as given in table 7.1. This program consists of the following code with initially sem = N-1.

```
P1:  if (REPADD (sem,-1) ≥ 0) then go to P3

P2:  REPADD (sem,1)
     go to P1

P3:  REPADD (sem,1)
     go to P1
```

Note that this code is equivalent to the incorrect PV-semaphore program of section 1; the only difference between these two programs is the initial value of sem. If the initial value of sem is independent of N, then the program possesses a compact REACHABILITY TREE.

REACHABILITY SET DESCRIPTION

S1 $n_1 + n_2 + n_3 = N$, $n_1 \geq 2$, sem = $n_1 - 1$
moves from P1 to P3 lead to *, if $n_1 \geq 2$
or lead to S2, if $n_1 = 1$
moves from P2 to P1 lead to *
moves from P3 to P1 lead to *

S2 $n_1 + n_2 + n_3 = N$, $n_1 \leq 1$, $n_1 + n_2 \geq 1$, sem = $n_1 - 1$
moves from P1 to P2 lead to *
moves from P2 to P1 lead to *, if $n_1 \leq 1$
or lead to S1, if $n_1 \geq 2$
moves from P3 to P1 lead to *, if $n_1 \leq 1$
or lead to S1, if $n_1 \geq 2$

Table 7.1

Strictly speaking we can not prove that the program can not have compact REACHABILITY TREE because we have not defined a REACHABILITY TREE. (Such a definition exceeds the range of the paper). A brief outline of the proof follows:

Imagine a parallel computer executing our abstract programs, which consist of (I) end (E) statements as defined in section 2. We suppose that each operation (including simultaneous Replace-f operations) is effected in a single cycle.

Let us call a program compact with respect to the given initial state s_0, if there is a time T_0 independent on N such that any state in $R(s_0)$ can be reached from s_0 within time T_0.

It can be shown that if a program has a compact REACHABILITY TREE then it is compact. It can also be shown that the program given above is not compact because the state $\{n_1 = 0, n_2 = N, n_3 = 0; sem = -1\}$ can not be reached from the initial state $\{n_1 = N, n_2 = n_3 = 0; sem = N-1\}$ in time asymptotically less than $\Omega(N)$.

Acknowledgement

The author thanks Allan Gottlieb, Robert Thau and Jim Wilson for proof reading and editing preliminary versions of this paper with great care.

References.

1.Gregory R. Andrews. Parallel Programs: Proofs, Principles, and Practice. Comm. ACM 24,3, pp.140-146 (March 1981).

2.E.A.Ashcroft. Proving Assertions about Parallel Programs. Journ.Comp.Syst.Siences 10, pp.110-135 (1975).

3.James E. Burns. Mutual Exclusion with Linear Waiting Using Binary Shared Variables. SIGACT News 10,2, pp.42-47 (Summer 1978).

4.E.M.Dijkstra. Hierarchial Orderings of Sequential Processes. Acta Informatica, Vol.1, pp.115-138 (1971).

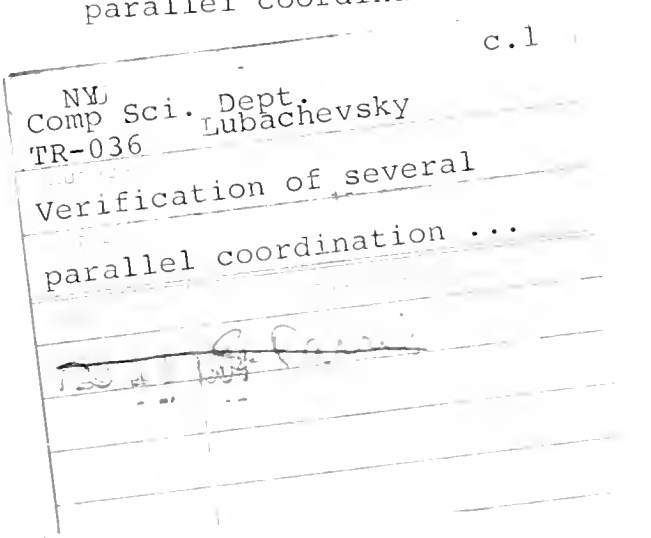
5.Allan Gottlieb, B.D.Lubachevsky, and Larry Rudolph. Basic Techniques For The Efficient Coordination of Very Large Numbers of Cooperating Sequential Processors. Courant Institute of Mathematical Sciences, Technical report No. 028, December 1980. Submitted to ACM Transactions on Program Languages and Systems. The abbreviated version titled "Coordinating Large Numbers of Processors" published in Proc. 1981 Intern. Conf. Parallel Processing, August 25-28, 1981.

6.David Korn. Scientific Code Conversion. Ultacomputer Note No.23, Courant Institute of Mathematical Sciences, New York University, 1981.

7.Susan Owicki and David Gries. Verifying Properties of Parallel Programs: An Axiomatic Approach. Comm. ACM 19, No.5, pp.279-285 (May 1976).

8.Susan Owicki and Leslie Lamport. Proving Liveness Properties of Concurrent Programs. (Unpublished).

NYU
Comp. Sci. Dept.
TR-036
Lubachevsky
Verification of several
parallel coordination ...



N.Y.U. Courant Institute of
Mathematical Sciences
251 Mercer St.
New York, N. Y. 10012

This book may be kept

FOURTEEN DAYS

A fine will be charged for each day the book is kept overtime.

FEB 14 1984

GAYLORD 142

PRINTED IN U.S.A.